



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Helmut Emmelmann
U.S. Patent App. No.: 09/449,021
Filed: November 24, 1999
Title: *Interactive Server Side Components*
Group Art Unit: 2192
Examiner: C. Kendall

APPELLANT'S BRIEF

CERTIFICATE OF MAILING

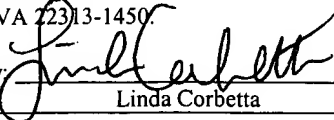
I hereby certify that this correspondence is being deposited as first class mail with the United States Postal Service on the indicated below, in an envelope addressed to:

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Date:

2/8/06

By:



Linda Corbetta

APPELLANT’S BRIEF

TABLE OF CONTENTS

I.	REAL PARTY IN INTEREST	1
II.	RELATED APPEALS AND INTERFERENCES.....	1
III.	STATUS OF CLAIMS	1
IV.	STATUS OF AMENDMENTS	1
V.	SUMMARY OF CLAIMED SUBJECT MATTER	1
VI.	GROUND OF REJECTION TO BE REVIEWED ON APPEAL	3
VII.	ARGUMENTS.....	3
A.	THE EXAMINER IS MISTAKEN REGARDING THE NATURE AND SCOPE OF THE CITED PRIOR ART	3
	1. <u>The Combination of Truong and D'Arlach Is Improper</u>	3
	2. <u>The Examiner Misinterprets D'Arlach</u>	5
	3. <u>The Examiner Has Misconstrued Applicant's Components</u>	6
B.	CLAIMS 1-8, 28, 29, 41, 42, 51-66, 68-96, AND 114-128 ARE PATENTABLE UNDER SECTION 103(A) OVER THE COMBINATION OF U.S. PATENT NO. 6,151,609 ("TRUONG") AND U.S. PATENT NO. 6,026,433 ("D'ARLACH").....	8
	1. <u>Claims 1, 59-63, and 68-73</u>	8
	2. <u>Claims 51-58, 74-89, 114-124 and 128</u>	13
	3. <u>Dependent Claims 2-5, 64-66 and 94-95</u>	23
	4. <u>Claims 6-8</u>	26
	5. <u>Claims 125-127</u>	29
	6. <u>Claims 90-93 and 96</u>	30
	7. <u>Dependent Claims 28 and 29</u>	33
	8. <u>Dependent Claims 41 and 42</u>	34
C.	CLAIMS 22-24, 26, 27, 30-33, 43, AND 67 ARE PATENTABLE UNDER SECTION 102(E) OVER TRUONG	34
	1. <u>Claim 26-33 and 43</u>	34
	2. <u>Claims 22-24</u>	38
	3. <u>Dependent Claim 67</u>	40

D.	CLAIM 25 IS PATENTABLE UNDER SECTION 103(A) OVER THE COMBINATION OF TRUONG, D'ARLACH AND U.S. PATENT NO. 6,651,108 TO POPP ET AL.	40
VIII.	CLAIMS APPENDIX.....	42
IX.	EVIDENCE APPENDIX.....	62
X.	RELATED PROCEEDINGS APPENDIX.....	63



APPELLANT'S BRIEF

I. REAL PARTY IN INTEREST

The sole inventor is applicant and the real party in interest.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

III. STATUS OF THE CLAIMS

Claims 1-8, 22-33, 41-43, 51-96 and 114-128 stand finally rejected and are subject of this appeal.

Claims 9-21, 34-40, 44-50 and 97-113 have been previously cancelled as drawn to non-elected subject matter.

IV. STATUS OF AMENDMENTS

Following the receipt of an Advisory Action, Appellant has filed a minor amendment to Claim 6 after final rejection, just weeks prior to filing this appeal brief, in an attempt to facilitate resolution of one well-defined disputed issue regarding a "*selected component*." The claims as currently pending (with proposed amendment shown) are reproduced in the Appendix.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Applicant has developed a unique editor for editing server-based **dynamic web applications**. A server-based **dynamic web application** is a software program that generates web pages upon request by a browser. In general, a user visiting a web site that is running a **dynamic web application** receives different versions of the same web page when viewing it multiple times. (page 1:16-18; Fig. 6). In contrast, a **static web site** is a set of documents that are delivered unchanged to the user's browser and which remain unchanged until changed by the author (page 1:15-16).

Applicant describes an embodiment that uses "page templates" and "components" to build server-based **dynamic web applications** (page 8:1-25). Instead of programming a web

application from scratch, the idea is to create the application by connecting preexisting software components (page 4:16-18 and page 5:7-12). Applicant's components are program modules or program classes containing instructions to generate browser code, and possibly instructions to react on user input (pages 8:28 - 9:7; pages 19:26-20:12). Applicant's page templates are used to specify how the components are to be connected to each other and to the layout (page 5:7-9 and page 8:1- 4). When the user visits the web application, page templates (26) and especially components (27) are executed (Fig. 7; page 8:28; page 16:14-18 (execution of components); and page 46:3-23 (execution of templates)) and thus transformed into the generated pages and sent to the browser (23) (page 15:3-5) for display to the user.

Advantageously, Applicant's components operate on the server yet can still interact with the user via multiple page requests (page 4:21-22; page 5:13-19; page 9:28 to page 10:4; pages 11:21-13:30). Thereby, the number and kind of components per generated page can change dynamically (page 4:23-28; page 11: 7-19). The preferred implementation of applicant's interactive server side components (ISSC) is described in section B on pages 14 to 24, and in Section D on pages 40 to 51, which discloses how the object oriented language heitml is used to implement page templates and ISSC's.

WYSIWYG editing of dynamic web applications is a complicated problem, because pages dynamically change during execution of the web application. Thus, there is no single page view that could be shown to a developer for editing. Applicant's invention addresses this problem by showing a running application in the editor (pages 3:27-4:2).

Advantageously, in applicant's editor the application looks similar to and functions like the normal running application as viewed by the end user (page 4:29-30; page 5:20-21; Fig. 1) (sample application as viewed by the end user) and Fig. 2 (applicant's editor). Conventional editing technology does not really address this issue – in general, page templates look different than the generated page as viewed by the user (pages 3:29-4:4). In contrast, applicant's invention has a special page generator (162) that runs the application while inserting editing features so that the editor shows the generated pages (163) to the developer (pages 4:29-5:4; pages 5:21-24; page 26:5-7; and Fig. 18) while actually applying changes to the page template (161) (page 5:25-26; page 26:10-13, for page generator, see pages 26:26-27:2, and Fig. 18; for editing features, see element 164 and section 5b on page 30, and section 5d on page 31). In addition, the editor comprises a client part in form of scripts for download into the browser (Fig. 18, element 165;

section 6 on page 32-36) to process user interactions and a server part for changing the page templates (element 166, section 7 on pages 36-40). Using applicant's editor would cause the display of an application, for example, a shopping application, to have a functional shopping basket thereby allowing the developer to add and/or remove items from the basket and to see the end user's actual view during editing.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

- i. Whether claims 1-8, 28, 29, 41, 42, 51-66, 68-96¹, and 114-127² are patentable under Section 103(a) over the combination of U.S. Patent No. 6,151,609 ("Truong") and U.S. Patent No. 6,026,433 ("D'Arlach").
- ii. Whether claims 22-24, 26, 27, 30-33, 43, and 67 are patentable under Section 102(e) over Truong.
- iii. Whether claim 25 is patentable under Section 103(a) over the combination of Truong, D'Arlach and U.S. Patent No. 6,651,108 to Popp et al.

VII. ARGUMENTS

Applicant's claims recite features that are not taught or suggested by the cited references, either alone or in combination, as discussed in sections B through D below. However, applicant will first discuss the cited art.

A. THE EXAMINER IS MISTAKEN REGARDING THE NATURE AND SCOPE OF THE CITED PRIOR ART

1. The Combination of Truong and D'Arlach Is Improper

In rejecting applicant's claims, the examiner has relied primarily on the editor disclosed in the Truong patent, either by itself or in combination with certain features disclosed in the D'Arlach patent. However, applicant has a fundamental disagreement with the Examiner regarding the nature, the scope, and the application of both of these prior patents to applicant's pending claims.

¹ The Examiner has not provided any explicit reasoning in the final rejection as to why Claim 73 was rejected. In earlier actions, Claim 73 had been rejected as obvious over the combination of Truong and D'Arlach, so applicant has discussed this claim on that basis in this section.

² The Examiner has not provided any explicit reasoning in the final rejection as to why Claim 128 was rejected.

The Truong patent discloses a plain text editor that, however, does not follow the WYSIWYG principle.³ (See Truong Fig. 5 and Col. 10:45-50). Thus, by itself, Truong does not teach or suggest the ability to **run** an application being edited while editing and the examiner has acknowledged this key fact. (See Final Rejection dated 6/6/05 at p. 4). Truong's editor also does not teach or suggest anything like applicant's components. However, the Examiner continues to rely upon Truong in forming his rejections of the claims, and this reliance is misplaced. Truong provides a conventional editor that acts on simple text files, and there is no teaching or suggestion, explicit or inherent, that it could transform or **run** these text files, or that it could edit components having executable code or that it would be desirable to modify Truong to include these features.

D'Arlach discloses an editor that follows the WYSIWYG principle, but can handle only static documents, and not server based dynamic web documents or web applications. Consequently it, like Truong, does not teach or suggest **running** an application being edited during editing. Significantly, the Examiner now finally admits that a claim to "dynamically editing a web page while the web page is being **run** on the browser" would be distinct. (See Advisory Action mailed 11/30/05). *This is one of the key principles underlying various of the claimed embodiments, and applicant continues to assert that this feature is adequately recited in several different ways in the pending claims.* For example, applicant's claimed embodiments contain various features which allow the editing of functional applications, including a "page generator" that **runs** the application during editing and at run-time, and "executable components" with which one can build server side applications. *In fact, web based editing of executable server side components is another important principle underlying the various claimed embodiments.* None of these features are taught or suggested by Truong or D'Arlach, either alone or in combination, nor is there any motivation described in those references to incorporate such features.

Applicant therefore submits that the combination is improper. The features of D'Arlach could not be readily incorporated into the editor of Truong, and likewise, features of Truong

³ Pronounced *WIZ-zee-wig*. Short for *What You See Is What You Get*. An application following the WYSIWYG principle allows the user to see on the display screen what will appear on the printed document. This should be contrasted with some word processors, for example, that cannot display different fonts and graphics on the display screen even though the formatting codes have been inserted into the file. (adopted from www.webopedia.com)

could not readily be incorporated into the editor of D'Arlach. Thus, applicant submits that the Examiner has failed to establish a *prima facie* case of obviousness.

2. The Examiner Misinterprets D'Arlach

Applicant had a fundamental disagreement with the Examiner regarding the nature and scope of the D'Arlach patent, because in the Final Rejection, the examiner applied D'Arlach in a way that assumes D'Arlach is capable of editing web applications. In the advisory action, however, the examiner advised that claiming "dynamically editing a web page while the web page is being run on the browser" would be distinct, so the examiner might have changed his mind. On the other hand, in the same advisory action, the examiner interprets a working copy of D'Arlach's templates as a functional and working duplicate of the template, which is possible only if D'Arlach's templates were programs. Applicant reasons below that D'Arlach's templates are plain documents and not programs, and consequently they can not be interpreted as functional or working.

D'Arlach teaches the use of web site "*templates*" to simplify web site development for the author. Instead of developing a complete web site anew, the author is given a template as a starting point. The user then customizes the template, and finally, publishes the result. The use of the words "Web Site" in D'Arlach's title suggest that D'Arlach's editor is intended for web sites, and not for server side dynamic web applications. Also, while D'Arlach's editor is itself a server side dynamic web application, this by no means suggests that its editor can edit such applications. According to D'Arlach, final generation of pages takes place when they are published, not at run time. (*See D'Arlach at col. 5:30-33*).

Server based dynamic web applications generate web pages dynamically every time a user visits the application. Web sites, on the other hand, are typically static and do not have generation taking place on the server while a user visits the site. *See, for example, D'Arlach at col. 5:30-33: "When the template is published as a Web site, the database is used to generate a set of Web pages that make up the new site."* It is thus clear that the generation in D'Arlach takes place when the web site is published, and not later when users visit the site. Thus, D'Arlach is simply not suited for developing dynamic server based web applications, and in fact, D'Arlach teaches away from editing a running application. In contrast, applicant's invention is

advantageous in that a user can edit server side web applications while the application is running such that it appears and functions normally.

The Examiner continues to assert that D'Arlach has pointed relevance because it provides for a "working copy," noting that

D'Arlach does disclose in analogous art, creating or editing a working copy of a user's site with the option of publishing the updated modified page or creating a new user web site. Therefore it would have been obvious . . . to combine Troung and D'Arlach because, it would enable a working copy or user web page to be modified dynamically.

(See Office Action dated 6/6/05 at p.4, *citing* D'Arlach at col. 5:15-25). The Examiner has improperly latched onto the term "working copy" as used in D'Arlach to support and justify his conclusion. However, the "working copy" described in D'Arlach is clearly not the "*functional application*" referred to in applicant's claims, and for this reason, the Examiner's logic must fail. A "working copy" means a copy for the user to work on, not a copy that works as a running and functional application. This means that the working copy is a static copy. This interpretation is supported by the disclosure of D'Arlach, which says that after changes to the web site are completed, the user may publish those changes. (See D'Arlach at col. 5:15-25):

*To allow creating a new Web site a CGI program first makes a copy of an exiting template in the server computer. The user then customizes or edits the **working** copy of the template, which is the user's site, through a series of forms displayed by the browser in the client computer. After making desired changes to the site, the user may publish it.*

The Examiner's misinterpretation is repeated in the Advisory Action dated 11/30/05, where he notes that "a working copy is a functional/working duplicate of the document being customized." However, D'Arlach does not disclose anywhere that the working copy is a running, functional application – this is solely a misinterpretation by the Examiner. Thus, it is clear to applicant that D'Arlach's editor is only useful for editing a static web site, not for editing a functional, running web application, and it does not provide any teaching or suggestion of any way to dynamically modify a functional, running web application, nor does it describe any motivation to do so.

3. The Examiner Has Misconstrued Applicant's Components

Applicant's invention allows the developer create web applications by plugging together components using the inventive editor. These special type of components contain instructions,

are executable on the server side, and generate browser code. In applicant's preferred embodiment, components are implemented as objects of an object oriented programming language.

Sometimes (*See* Office Action dated 6/6/05 at p.5, *citing* D'Arlach at col. 5:25-45) the examiner mistakenly interprets the "elements" described in the D'Arlach patent (D'Arlach at col. 4:62) as somehow equivalent to applicant's claimed components. However, the elements described in the D'Arlach patent do not contain instructions and are not executable, but are merely static data items stored in a database (*See* D'Arlach at col. 5:14-16). In fact, the database structure disclosed by D'Arlach (see col.5:1-15) shows that the database does not contain instructions for execution on the server nor does it contain executable components. The cited portion further discloses that D'Arlach's invention only has two kinds of elements: a text and a button element, and neither contains instructions for execution on the server. Further, the examiner does not even argue that D'Arlach's elements contain instructions or are executable.

Sometimes (*See* Office Action dated 6/6/05 at p.5 and p.6, *citing* Truong at col. 2:1-5) , however, the examiner also mistakenly interprets normal HTML elements such as fields and buttons as somehow equivalent to applicant's claimed components. However, HTML elements are executed on client side and do not generate browser code.

In the Advisory Action, the Examiner interprets executing components as somehow equivalent to certain actions inside D'Arlach's editor, stating that "D'Arlach shows the user being able to interactively input data through the browser with regards to processing New html, which when processed (executed) is displayed back to the user. Executing as claimed [by] applicant is equivalent to modifying a template and displaying it on the browser." Thus, the examiner mistakenly interprets the part of D'Arlach's editor that performs these actions as somehow equivalent to a component. Applicant's specific components are, however, intended for reuse inside applications developed with applicant's editor. So, applicant's claims typically contain additional restrictions that require components on page templates, or that require the editor to support specific editing functions for operating on the components on document templates. The editor part cited by the examiner does not fulfill this additional restriction. D'Arlach by no means teaches or suggests the use of the cited editor part inside web site templates.

Thus, applicant submits that the Examiner either has misinterpreted and misconstrued the teachings in the cited references or simply did not take into account all the restrictions on components in the claim language.

B. CLAIMS 1-8, 28, 29, 41, 42, 51-66, 68-96, AND 114-127 ARE PATENTABLE OVER THE COMBINATION OF TRUONG AND D'ARLACH

1. Claims 1, 59-64, and 68-73

Independent Claims 1 and 59, and claims dependent from claim 59, form the basis for a first group considered to stand or fall together, and not with other claims or groups.

Claims 1 and 59 are each independent claims that are directed to a basic two-part structure that runs a web application and an editor. For example, Claim 1 includes "*a page generator*" and "*an editor*," while Claim 59 includes "*a document generator*" and "*an editor program*." The *page generator* and *document generator* elements recited in these claims are similar and because they require running the web application, and this provides a distinct basis for concluding that these claims are patentable over the cited art. Other groups of claims are focused on different features, such as interactive software components (claims 6-8); or editing software components (claims 2-5, 51-58, 64-66, 74-89, 94-95, 114-124 and 128); or scripts for download into client (claims 41-42, 90-93, 96); or handles (claims 28, 29); or a method for editing an application (claims 125-127).

Claim 1 is considered patentable over the cited combination because (1) the recited page generator runs the application, and (2) the recited editor interacts with features incorporated onto pages being displayed in the web browser. This is recited in Claim 1 as:

a page generator running one of the applications being developed by generating the generated documents including additional editing features for interpretation by the browser program;

an editor directly operating on the pages displayed by the browser via the editing features, thereby allowing the user to work on a functional application during development;

Neither Truong nor D'Arlach make any teaching or suggestion of editing an application while running it, nor is there any suggestion or motivation that it would be desirable to do so. For that reason alone, Claim 1 is patentable over the proposed combination.

Further, the Examiner has acknowledged that “Truong doesn't explicitly disclose an editor capable of directly operating on the documents displayed by the browser thereby allowing the user to work on a functional application during development.” (See Office Action dated 8/15/2005 at p. 4). However, the Examiner also has stated that it would be distinct if applicant claimed “dynamically editing a web page while the webpage is being run by the browser.” (See Advisory Action dated 11/30/05 at continuation sheet) Applicant’s claim provides exactly that! The Examiner’s erroneous evaluation of this claim is belied by his statement in the Advisory Action that Claim 1 only calls for “editing a page being displayed by the browser.” (*Id.*) The phrase “via the editing features” is, however, a key structural limitation of the claim language that was apparently not considered by the Examiner, and which connects the running application to the editing process. The claim recites that the editing features are part of the generated documents, and the preamble states that the generated documents are displayed by the browser. The editor is then coupled via the editing features, and *voila*, you are editing a running application in the browser.

Claim 1 is thus directed to a web based editor that can handle and interact with editing features incorporated into pages displayed in the browser as part of a running application. The Examiner continues to assert that the creating or editing of a working copy as disclosed in D’Arlach is somehow the same as the editor claimed by applicant, *citing* D’Arlach at 5:15-25. (See Office Action dated 8/15/2005 at p. 4; *see also* Advisory Action dated 11/30/2005 at continuation sheet). Applicant submits that the Examiner is simply mistaken in his conclusion – the “working copy” referred to in D’Arlach is not a running application, but simply a static, redundant copy – one to mark up, scratch up, or otherwise tinker with until revisions are complete – and then, and only then, will the updated material be **published**.

The Examiner has focused on the claim language “thereby allowing the user to work on a functional application during development” in concluding that the claim is only directed to an editor capable of working on a displayed page. However, that fails to consider the full scope of the physical structure as actually described in applicant’s claims. For example, in Claim 1, the preamble states that an application (not restricted as to location) responds to a browser request by generating documents and sending them to the browser for display. This is a conventional page generator function. A recited element of the claim, however, is a “page generator” that runs the application generating generated documents with additional editing features. Another recited

element is an editor that operates on the pages via the editing features. These structural limitations clearly correlate the functional aspects to the structure.

The cited portion of D'Arlach (column 5:15-25) does not teach or suggest that during editing, pages should function similar to the actual pages that are displayed in the browser, nor that pages are functional at all during editing. In contrast, applicant's claim requires that the application be running, and this allows it to remain functional during development.

The Examiner has stated that Truong discloses a page generator running an application being developed and sending generated documents to the browser. (Final Rejection dated 6/6/05 at p. 4). However, a thorough review of the Truong patent reveals that its page generator runs only the editor, not the application being developed. The Examiner relies on Truong at 10:45-50, but this portion explicitly states that the text of the file being edited is sent directly to the browser, without actually executing the file. Also, Fig. 5 of Truong shows the source code of a script in the editor, which indicates that the file being edited is not run during editing.

Further, the templates which are edited in D'Arlach are static web sites, and consequently can not be executed. Applicant has repeatedly argued that D'Arlach does not edit dynamic web applications (*see* section A.2). However, the Examiner mistakenly disagrees. The Examiner cites column 5:15-21 as disclosing that D'Arlach can edit a working copy of the template (which is the user's site) through forms displayed in the browser. The Examiner interprets this passage as supporting having the ability to edit dynamically. Applicant submits that being able to edit web sites dynamically does not teach or suggest editing web applications. In fact, the use of the term "web site" suggests in itself that D'Arlach's editor can edit static web pages, but not applications (like database searches or shops).

Thus, Claim 1 is patentable over the cited combination.

Independent Claim 59 also stands rejected based on the combination of Truong and D'Arlach. However, for all the same reasons as claim 1, Claim 59 is also patentable over the cited combination. Similar to Claim 1, Claim 59 is an independent claim directed to a software development system for dynamic web documents capable of displaying functional applications during editing. In the Final Action, the Examiner referred to the reasoning for claim 6, but the reasoning for claim 6 is wholly inapplicable to claim 59. Claim 59 recites an editor, whereas Claim 6 does not; Claim 6 recites components, whereas Claim 59 does not.

As explained above, Truong discloses an editor that works on plain text documents. Consequently, when Truong's editor is used on a dynamic web document, it shows just the source code, does not follow the WYSIWYG principle, and does not transform the dynamic document during editing. In contrast, the inventive editor is capable of transforming a dynamic web document being developed during editing and so the dynamic web document appears functional during editing. This distinction is expressed by the following claim language, "generated documents ... which look and function similar to the end user's view of the documents" and by "the editor program comprising first instructions for requesting the document generator to process a dynamic web document leading to a generated document." This language makes clear that the editor indeed transforms the dynamic web document during editing. Additional claim language, namely "*instructions to modify **the dynamic web document***" make clear that the dynamic web document that is being edited is the one being requested.

As noted many times, D'Arlach's editor can not edit dynamic web documents, but is used for static documents only. In contrast, claim 59 requires "*A software development system for developing dynamic web documents*" and clearly defines the term using the claim language "said dynamic web documents operating by being transformed into an end user's view upon a request by a web browser, the end user's view being provided to the browser in response to the request." In contrast, document generation in D'Arlach happens during publishing, not upon a request by a web browser, whereas dynamic web documents are generated when users visit them. Thus, Claim 59 is patentable.

Claims 60-73⁴ are dependent from Claim 59, and for all the same reasons, applicant submits that Claims 60-73 are likewise not taught or suggested by Truong, either alone or in combination with D'Arlach.

With regard to claim 61, the examiner cited Truong (Col. 8:39-50) as disclosing further instructions for execution during document generation to collect edit information for use by the editor. The cited portion describes document generation as part of the editor itself. According to claim 59, "the document generator" means the document generation usually required for display of a dynamic web document. In contrast, the cited portion of Truong is concerned with document generation taking place inside the editor.

⁴ Claim 67 was placed by the Examiner in another grouping, but applicant believes this claim is properly considered here with its base claim.

With regard to Claim 63, the examiner states that Truong does not explicitly disclose automatically requesting that the document generator process the dynamic web document if required. However, the examiner states that D'Arlach discloses analogous art, citing item 620. D'Arlach describes a CGI program that transmits the updated page (see Col. 6:8-20). Since D'Arlach's method does not repeat the original request, URL parameters of the original request for the page get lost. Therefore, D'Arlach's method does not work in general for dynamic web documents. Claim 59, however, requires the editor to work for dynamic web documents.

With regard to claim 68, the examiner referred to Claim 61 for his reasoning, and applicant likewise refers to his response to that reasoning, as discussed above.

Claim 69 is dependent on 68, and for all the same reasons, applicant submits that Claim 69 is likewise not taught or suggested by Truong, either alone or in combination with D'Arlach. With regard to claim 69, the examiner cited Truong at (Col. 3:35-38) as disclosing that the editor uses the edit information to modify the dynamic web document. However, the cited portion refers to an editor selection form used to select a file for editing, not to modifying a dynamic web document or using edit information.

Claim 70 is dependent on 69, and for all the same reasons, applicant submits that Claim 70 is likewise not taught or suggested by Truong, either alone or in combination with D'Arlach. With regard to claim 70, the examiner referred to Claim 64 for reasoning. Claim 70 requires "*position information on the components contained in the document template*" while claim 64 does not. Applicant therefore submits that Truong does not teach or disclose position information and therefore claim 70 is patentable.

Claim 72 is dependent on 71, and for all the same reasons, applicant submits that Claim 72 is likewise not taught or suggested by Truong, either alone or in combination with D'Arlach. With regard to claim 72, the examiner cited Truong (Col. 7:59-67) as disclosing initiating a reload in the browser. However, the cited portion discloses loading of the remote editor program, does not initiating a reload in the browser. A reload in the browser means that the browser will load the same page a **second time**. In contrast, the cited portion discloses a **first time** load.

With regard to claim 73 the examiner cites D'Arlach (Col. 10:15-25). However, the cited portion deals with managing complete web sites and not with elements of web pages. (See Col. 6:4-6) D'Arlach's editor requires a server interaction in order to display information on an

element. In contrast, claim 73 requires information on an element to be displayed without requesting the document generator.

2. Claims 51-58, 74-89, 114-124 and 128

Independent Claims 51, 74, and 114 and their related dependent claims (52-58, 75-88, 115-124, 128) form a second group considered to stand or fall together, and not with other claims or groups. Among other things, these claims deal with editing components, and they all recite “a plurality of components” element, wherein the components are limited to include “instructions to generate browser code.” This is a key structural limitation that distinguishes components from the static database element.

Claim 51 is an independent claim describing a system for editing components on web document templates and stands rejected as obvious based on the combination of Truong and D’Arlach. In the Final Rejection, the Examiner simply pointed to the reasoning applied to claim 6. With regard to Claim 6, the Examiner cited Truong (col. 2:1–5) as disclosing a plurality of components. The cited portion of Truong, however, refers to browser built-in HTML elements, like forms and fields. However, these elements do not generate browser code and can therefore not be interpreted as the “components” recited in applicant’s claims.

As discussed in section A.3 above, D’Arlachs “elements” cannot be interpreted as equivalent to the claimed components since the claim requires “*components containing instructions to generate browser code.*” In the advisory action, the Examiner interpreted a specific part of D’Arlachs editor being equivalent to a component. Claim 51, however, requires “*a user interface for editing functions used for maintaining components on document templates.*” However, no such part can possibly be contained on D’Arlachs templates (see col. 4:60 to col. 5:9), and D’Arlach does not disclose any editing functions for maintaining this part on a site template.

In fact, the cited references do not teach or suggest a user interface having editing functions for maintaining components. The editing functions actually disclosed in Truong are clearly identified as working on text and not on components by stating: “the text may be edited at the web browser using editing features.” (See Truong at col. 10:47-50) In addition, even though Fig. 5 of Truong shows a user interface for the editor, it does not show “*a user interface for editing functions used for maintaining components on document templates,*” as recited in claim

51. D'Arlach describes functions for modifying elements on a template. However, as already discussed, these passive elements are not active components containing instructions to generate browser code.

Claims 52-58 depend from Claim 51, and for all the same reasons, these claims are patentable over the cited combination.

With regard to claim 52, the Examiner cited Truong column 7 line 1 to 15 as disclosing components including fourth program instructions including steps to generate browser code prior to transmission to the browser program. The cited portion describes web browsers that can execute scripts. Scripts can indeed be used to generate browser code, but inside the browser. The Examiner cited Truong at column 2 line 1 to 5 as disclosing components, identifying HTML textboxes and buttons as components. However, HTML text boxes and buttons do not contain scripts for generation of browser code. In addition, HTML element and scripts are clearly part of the browser. In contrast, the claim explicitly requires the generation of browser code prior to the transmission to the first software program. D'Arlach's elements are data items and therefore do not contain instruction including steps to generate browser code.

Claims 53-58 depend from Claim 52, and for all the same reasons, these claims are patentable over the cited combination.

Claim 55 is directed to edit functions working on components. The Examiner refers to his reasoning towards claim 6. However, the portions of D'Arlach cited by the Examiner do not teach or suggest the operations of adding a component or removing a component from a document template as required by the claim.

With respect to claim 56, the Examiner cites D'Arlach's figure 5 number 506. According to column 5 lines 38 to 40, step 506 deals with selecting a site template, not with elements or components. Claim 56, however, requires the calling of fourth program instructions. Fourth program instructions are introduced in claim 52 as instructions included inside at least some components to generate browser code prior to transmission to the first software program. As explained above, D'Arlach's elements are merely data items not containing any instructions. In contrast, claim 56 requires instructions included in some components to be called.

Claims 57-58 depend from Claim 56, and for all the same reasons, these claims are patentable over the cited combination.

With respect to claim 58, the Examiner cited D'Arlach column 10 lines 15 to 25 as disclosing clicking on the generated document. The cited portion of D'Arlach, however, seems to disclose clicking various buttons in order to manage complete web sites. The term "the generated document" used in claim 58 refers back to claim 56, saying that the generated document is generated from a document template. The menu buttons cited by the Examiner are part of the editor and are not generated from the template.

Claim 74 is an independent claim describing a software development system for document templates and stands rejected as obvious based on the combination of Truong and D'Arlach. In the Final Rejection, the Examiner simply pointed to the reasoning applied to claim 6, i.e., that Truong (col. 2:1-5) discloses a plurality of components. The cited portion refers to browser built-in HTML elements, like forms and fields. These elements do not generate browser code, however, and can therefore not be interpreted as the "components" recited in applicant's claim.

As discussed in section A.3, D'Arlach's "elements" can not be interpreted as equivalent to components since the claim requires "*components, that include instructions to generate browser code for transmission to the first software program.*" In the advisory action, the Examiner interpreted a specific part of D'Arlach's editor as being equivalent to a component. Claim 74, however, requires "*edit functions maintaining components on document templates.*" D'Arlach's templates can not possibly contain such a part, see col. 4:60 to col. 5:9, and D'Arlach does not disclose any editing functions maintaining such a part on a site template.

In fact, the cited references do not teach or suggest editing functions for maintaining components. The editing functions actually disclosed in Truong are clearly identified as working on text and not on components, for example: "*the text may be edited at the web browser using editing features.*" (See Truong at col. 10:47-50). D'Arlach has functions to modify D'Arlach elements on a template; however, these elements are not active components containing instructions to generate browser code, as already discussed.

The claim explicitly requires document templates to be dynamic, stating that the set of components on the generated documents can vary for different document requests for the same document template. D'Arlach creates web sites, in contrast to web applications, and generates pages while publishing, not per request, as explained above. This means that according to D'Arlach, a page can change only by modifying the template and publishing it again.

Claims 75-89 depend from claim 74, and for all the same reasons, should be considered patentable.

With regard to claim 75, the examiner cited D'Arlach (Col. 10:10-25 and 50-60). The cited portions refer to managing web sites and web pages, and not to components as required by the claim. In addition, the cited portions of D'Arlach do not reveal the operations of adding a component to a document template as required by the claim.

With regard to claim 76, the examiner cited Truong (Col. 6:57-63) as disclosing that tag syntax is used to denote components on document templates. The cited portion reveals only that HTML tags are denoted using tag syntax on document templates. However, HTML tags are not "components" as recited in applicant's claims since, according to claim 74, components generate browser code prior to transmission to the first software program, and also, components cooperate with the editor program (See also section A.3). This is not the case for HTML tags. On page 17 of the Final Rejection with regards to claim 76, the examiner argues that applicant has merely claimed in his limitations "wherein tag syntax is used to denote at least one component." However, the use of component refers back to the use of component in claim 74 on which claim 76 depends. Claim 74 states "a plurality of components, that include instructions to generate browser code for transmission to the first software program." Applicant submits that the examiner is wrong in considering only the limitation in claim 76 for components. Applicant further submits that the limitation in claim 74 does not allow components to be interpreted as HTML tags, because HTML tags do not include instruction to generate browser code for transmission to the first software program.

With regard to claim 78, the examiner cites D'Arlach (Col. 10:10-35) as disclosing excluding a component that can react interactively on subsequent document requests from the generated document. As discussed with regard to claim 74, D'Arlach's elements can not be interpreted as components because they are merely data elements, not software components, and do not contain instructions. In particular, a component that can react interactively as required by claim 78 can not be interpreted as an element, because elements are data items that can not react. Further, the cited portion talks about editing the web site template and thereby removing an element. The claim, however, refers to a web application that, for selected requests, excludes components, e.g., a shopping application that hides a shopping basket in case it is empty. The claim language expresses this by the use of "the generated document" referring back to claim 74.

With regard to claim 79, the examiner cites Truong (Figure 3b, item 164, and related text). The cited portion refers to handling of the ABORT button of the editor. The claim, however, deals with the components on the document templates created with the document development system. Therefore, the cited portion is irrelevant to this claim.

With regard to claim 80, the examiner cite D'Arlach (Col. 5:23-25) and "storing of the customized template limitation." Claim 80 is dependent on claim 79, and as reasoned with regard to claim 78, "excluding components" refers to excluding while an application is running without changing the template. In addition, there is no teaching of session memory or of any information stored in session memory.

With regard to claim 81, the examiner cited D'Arlach (Col. 4:60-65). The cited portion deals with manually editing a web site template. The claim, however, refers to a web application that, for selected requests, excludes certain components, e.g., a shopping application that hides a shopping basket in case it is empty. The claim language expresses this by the use of "the generated document" referring back to claim 74. In addition, the cited portion does not teach or suggest anything like a first component containing sixth instructions to decide about exclusion. In fact, D'Arlach's elements are data items and do not contain instructions.

With regard to claim 82, the examiner cited Truong (Col. 3:30-35) as disclosing an editor taking the varying set of components into account. However, the cited portion describes the editor input and editor selection form used to select a file for editing, and does not seem to provide any information as to how the editor shows a particular file. However, Truong's Figure 5 clearly indicates that the editor shows the source code of an application being edited. In contrast, the claim requires the editor be able to take the varying set of components into account, which implies that the editor must be capable of showing various views of the same document.

With regard to claim 83, the examiner cited D'Arlach (Col. 4:60-65). The cited portion deals with manually excluding elements from a page template. The claim, however, refers to editing a web application that, for selected requests, excludes certain components. In applicant's editor, an application appears functional during editing and can include or exclude components while functioning. In contrast, D'Arlach's editor is for static web sites, and pages change only when they are manually changed by the author. The claim language expresses this by requiring an editable view that includes and excludes components similar to the end user's view of the document. In contrast, D'Arlach's editor just shows a static web page that changes only if the

user asks the editor to modify the template. On page 17 of the Final Rejection with regards to claim 83, the examiner states that applicant is simply rehashing arguments which have been previously discussed. However, claim 83 was significantly amended, and the arguments are highly relevant.

With regard to claim 84, the examiner referred to claim 4 for his reasoning. However, claim 4 is concerned with nested components. In contrast, claim 84 is concerned with the set of components on the document template and on the generated document. Since claim 5 is concerned with the set of components, applicant submits that for all the reasons given with regard to claim 5, Claim 84 is likewise not taught or suggested by Truong.

With regard to claim 85, the examiner cites D'Arlach (Col. 6:35-45). The cited portion deals with changing options from the set of available options inside the editor. In contrast, the claim requires the generated document to contain multiple instances of a component contained in the template. For example, in applicant's invention, a shopping basket component has multiple instances and could contain multiple products, depending on the number of articles actually placed into the basket. The use of the term "the generated document" refers back to claim 74.

With regard to claim 86, the examiner cited Truong (Col. 8:20-35) as disclosing instructions to qualify names generated into the browser code with the unique identifier. The cited portion seems to disclose including file names into a generated page. However, the file names are known before the generation process. In contrast, claim 86 requires that a unique identifier be assigned to each component instance, and that this unique identifier be used to qualify the names.

With regard to claim 87, the examiner cites D'Arlach (Col. 4:40-35). However, the cited portion describes the basic operation of D'Arlachs editor as a CGI program processing page requests. The cited portion does not seem to disclose elements, components, instances of components, nested components or templates. Instead, the cited portion deals only with the operation of the editor, but not with the operation of the site templates. In contrast, the claim requires that the software development system work on templates that, for selected requests, display multiple instances of a component in the template on the generated page. The claim language expresses this concept by the use of "the generated documents" referring back to claim 74.

With regard to claim 88, the examiner refers to his reasoning of claim 85, and applicant likewise incorporates his response to claim 85. In addition, applicant submits that D'Arlach does not have an editable view that displays multiple instances of a component, but instead, D'Arlach just shows a static web page that changes only if the user asks the editor to modify the template.

With regard to claim 89, the examiner cites D'Arlach (Figure 6, item 604). However, the cited portion refers to an element editing form and does not show instructions contained in components for generating browser code, as required by the claim.

Claim 114 is an independent claim describing a system for displaying dynamically generated documents, and the claim stands rejected as obvious based on the combination of Truong and D'Arlach. The examiner cites Truong (col. 6:30-35) as disclosing a plurality of components on the server, at least one of the components including first features to cooperate with an editor in editing said component, and second program instructions to generate browser code. The cited portion of Truong reveals hardware components of the client computer, but these components are not present on the server. In contrast, claim 114 requires a plurality of components on the server. In addition, the components referenced by Truong do not generate browser code. In contrast, claim 114 requires components to include second program instructions to generate browser code.

The examiner cites Fig. 3A of Truong, which shows a network editor page, as disclosing third program instructions on the server for generating generated documents for transfer to the client computer, thereby calling second program instructions of selected components. However, Fig. 3A of Truong shows various instructions to generate the network editor web page, but no instructions to call second instructions of components. In fact, the hardware components cited as prior art by the examiner do not have second instructions at all and therefore it is impossible to call them.

As discussed in section A.3 above, D'Arlach's "elements" can not be interpreted as equivalent to components since the claim requires "*at least one of the components including ... second program instructions to generate browser code.*" Also, HTML elements can not be considered equivalent to the claimed components as reasoned in section A.3 because of the claim language "*components for execution on the server.*" Also, despite the Examiner's comments to the contrary in the advisory action, a part of the editor can not be interpreted as a component because the part lacks first features to cooperate with the editor in editing this part itself as

required by the claim language “*components including first features to cooperate with an editor in editing said component.*” In addition, the use of the word cooperate makes clear that the components are not part of the editor.

Claims 115-124 and 128 depend from claim 114, and for all the same reasons, should be considered patentable.

With regard to claim 115, the examiner cited Truong (col. 10:55-59) as disclosing that first features include instructions for passing information to the editor. According to claim 114, components for execution on the server include first features for cooperation with the editor. The cited portion of Truong reveals the use of instructions for sending a complete file back to the server. These instructions are a part of the browser for execution on the client computer. In contrast, claim 115 requires instructions to be part of the components for execution on the server computer. On page 18 of the Final Rejection, with respect to claim 115, the examiner states that claim 15 [sic] does not claim instructions to be part of the components. Applicant disagrees. Claim 115 states “*first features include fourth program instructions*” and depends on claim 114 that states “*at least one of the components including first features.*” Since at least one component includes first features, and first features include program instructions, therefore, at least one component includes program instructions. Applicant therefore submits that the examiner is in error and that claim 115 together with its base claims requires instructions part of at least one component.

Claims 116-118 depend from Claim 115, and for all the same reasons, these claims are patentable over the cited combination.

With regard to claim 116, the examiner cited Truong (col. 7:60-67) as disclosing that part of the information is collected during execution of the components on the server. However, the cited portion describes execution of the editor, not of the components. According to the base claim 114, the editor cooperates with the components. This makes clear that the components and the editor are distinct items that cooperate. Truong does not teach or suggest any execution of components on the server. The examiner might have interpreted HTML text boxes and buttons (col. 2:1-5) as “components” in his discussion of claim 2. These HTML elements are part of the browser, however, and are not executed on the server. In contrast, claim 116 requires components to be executed on the server thereby collecting information for the editor. On page 18 of the Final Rejection, with respect to claim 116, the examiner states “*applicant argues that*

the contents are not executed on the server.” Applicant said nothing about “*contents*” and therefore assumes that the examiner actually meant to say “*components*.” In fact, applicant argues that Truong at col. 7:60-67 describes execution of the editor, not of the components. On page 18 of the Final Rejection, the examiner further states “*see claim 116, above which shows the remote editor which is loaded on the server,*” which seems to agree with applicants reading of col. 7:60-67 thus, while the examiner cited art to show execution of the editor, he did not cite art to show execution of the components. The claim, however, requires execution of selected components on the server.

With regard to claim 118, the examiner cited Truong’s Figure 5 as disclosing that the information includes attributes of the component. Figure 5 of Truong shows the user interface for the editor and does not provide any details about “the information” and therefore appears irrelevant. On page 18 of the Final Rejection, with respect to claim 118, the examiner states “*Applicant claims does not exclude editing features being indicated inside the browser, therefore Applicants argument is moot.*” Claim 118 and its base claims do not refer to editing features, and applicant’s reasoning does not refer to editing features. Fig. 5 of Truong does not show any details about “the information,” and it especially does not show that the information includes attributes of the component. In contrast, the claim requires “*said information includes attributes.*”

With regard to claim 119, the examiner cited Truong (col. 10:45-50) as disclosing first features including fifth instructions that display additional editing features. The cited portion seems to reveal editing features as part of the editor and browser. In contrast, the claim requires first features to include fifth instructions to display the editing features, whereby the first features need to be included in the component according to claim 114, not inside the editor or the browser. On page 18 of the Final Rejection, with respect to claim 118, the examiner included arguments about “editing features.” Since neither claim 118 nor applicants reasoning mentions editing features, the examiner’s argument appears misplaced. The examiner argues “Applicants claims does not exclude editing features being indicated inside the browser, therefore Applicants argument is moot.” Where the editing features are **indicated** does not seem to be relevant, but where the instructions that display the editing features are located is relevant. The claim requires “*first features include fifth instructions that display the editing features.*” The base claim 114 requires “*components including first features.*” Consequently, fifth instructions are claimed as

part of the components. The examiner did not cite prior art for the fifth instructions. The cited portion (col. 10:45-50) only talks about the editor and does not seem to give any features of the components.

Claim 120 depends from Claim 119, and for all the same reasons, this claim is patentable over the cited combination. With regard to claim 120, the examiner cites Truong item 602 and related text as disclosing handles. However, the cited portion talks about selecting an element but does not specify how, nor does it mention anything like a handle. D'Arlach does not disclose handles, but instead, elements are selected by directly clicking on them. As described in col. 9:13-14 of D'Arlach, clicking on the "Phone Services" button itself, and not a handle, brings up the "Elements Properties" screen. In addition, Fig. 10 does not show anything like a handle. In contrast, applicant's claim explicitly requires handles. In his arguments on page 18 of the Final Rejection, the examiner refers to his interpretation of handles on page 17 with respect to claims 28 and 29. Applicant likewise refers back to his discussion of these arguments. Applicant submits that neither D'Arlach or Truong show handles in the interpretation given by the examiner.

With regard to claim 121, the examiner cited Truong (col. 10:45-50) as disclosing first features including extensions for use by the editor. The cited portions reveal various features as part of the editor and browser. In contrast, claim 121 requires the first features to include the extensions, whereby the first features are included in the components according to claim 114, not inside the editor or the browser. With respect to claim 121 the examiner refers to his interpretation of extension using "as recited above." However, applicant does not find such an interpretation anywhere above in the Final Rejection, so applicant relies on the interpretation given on page 19 of the Final Rejection. There, the examiner argues that the applicant failed to provide further limitations to distinguish his extension from the prior art. Applicant submits that according to claim 121 "*first features include an extension for use by the editor*" and according to the base claim 114 "*at least one of the components including first features*" and therefore the claim language correctly limits the extension to be part of the components and not of the editor. The features given by the examiner are part of the editor and not of the components. Applicant therefore submits that the features given by the examiner can not be interpreted as said extension in the context of the claim.

Claim 122 depends from Claim 121, and for all the same reasons, this claim is patentable over the cited combination. With regard to claim 122, the examiner cited Truong (col. 8:65-67) as disclosing a web page for editing component attributes. The cited portion discusses the generation of web pages in general and does not disclose the generation of a web page for a specific purpose. In contrast, claim 122 specifically requires a page for editing component attributes. On page 19 of the Final Rejection, with respect to claim 122, the examiner states that he believes step 108 of Fig. 3A shows a page for editing component attributes. A review of col. 9:19-25 reveals, however, that step 108 shows the network editor web page that provides input fields for logon ID, password input and remote editor path. Therefore, the network editor web page in step 108 is not used for editing any attributes, but for a log in procedure. Claim 122 explicitly requires “*a page for editing the components attributes values.*”

With regard to claim 123, the examiner cited Truong (col. 6:57-63) as disclosing components denoted on document templates using tag syntax. However, the cited portion discloses **HTML tags** denoted on document templates using tag syntax. **HTML tags** are different from **components** as used by applicant in claim 114, since components are required to run on the server and generate browser code. Applicant submits that the cited portion does not disclose components denoted on document templates using tag syntax. With respect to claim 123, the examiner states on page 19 of the Final Rejection that the argument has already been discussed above and probably refers to his arguments on claim 76. Applicant refers to his answer, and in addition, notes that claim 123 and claim 76 have different base claims.

With regard to claim 124, the examiner cited Truong column 8 lines 65 to 67 as disclosing components including instructions to generate browser code. The cited portion discloses instructions to generate browser code, but it does not specify where. Claim 124 requires that these instructions be part of the components.

With regard to claim 128, applicant notes that the Examiner failed to provide any basis for rejection.

3. Dependent Claims 2-5, 64-65, and 94-95

Claims 2-5 are dependent from Claim 1 and should be considered patentable for all the same reasons. Claims 64-66 are dependent from claim 59 and should be considered patentable for the same reasons. Also, claims 94 and 95 are dependent from claim 50 and should be

considered patentable for the same reasons. However, each of these claims also add the notion of components, and therefore should be considered separately and apart from the other groups.

With regard to claim 2, the Examiner cites Truong (Col. 2:1-5) as disclosing components, giving text boxes and buttons as examples. Column 2 also mentions HTML elements that are part of the browser. Applicant submits that it is not possible to interpret these elements to be the same as the recited components, because claim 2 requires the components to be executed by the page generator, and the page generator does not execute HTML elements (See section A.3).

The Examiner further cites Truong (Col. 10:47-50) as disclosing features to insert, modify and delete a component. However, the cited portion introduces insert, modify and delete operations for characters, not for components.

The examiner acknowledges that “*Truong does not explicitly disclose wherein [sic] developed applications comprising document templates or editing components on templates and executing components on page templates.*” (See Final Rejection dated 6/6/05 at p.5) However, the Examiner asserts that D’Arlach discloses analogous art, citing (Col. 5:25-45). As previously described, D’Arlach’s templates are generated at publishing time (see Col. 5:30-33), while claim 1 requires templates to be generated upon each browser request. In addition, the objects and elements disclosed in D’Arlach are data items stored in a database and can not be executed. (see Col. 5:14-16). In contrast, claim 2 requires components to be executed.

The Examiner states that it would have been obvious to combine Truong and D’Arlach because using templates would enable users to create and maintain web sites easily and efficiently, and he cites D’Arlach at Col. 4:60-63. In contrast to D’Arlach’s templates, applicant’s templates are primarily used to create dynamic web applications and not merely to simplify development of static web sites.

Claims 3-5 are dependent on Claim 2, and for all the same reasons, applicant submits that claim 3-5 are not taught or suggested by Truong, either alone or in combination with D’Arlach.

With regard to claim 3, the examiner cited Truong (Col. 2:1-5) as disclosing that at least one of the components reacts interactively on user input by executing instructions on the server. However, the cited portion actually describes HTML elements that are capable of **sending** information to the server. Thus, it appears that the examiner believes HTML elements are relevant prior art for applicant’s components. However, claim 3 requires that the component themselves react and execute instructions on the server. In Truong, the HTML elements are

interpreted by the browser, but are not actually present on the server, which also means that they cannot execute instructions on the server. Thus, HTML elements can not be interpreted as components in the sense of claim 3.

On page 16 of the Final Rejection, the examiner argues that applicant did not disclose HTML elements interpreted by the browser as being present on the server, and therefore applicant's argument was moot as directed to an unclaimed feature. In fact, Applicant disclosed and claimed components containing instructions and executing said instructions on the server, using the claim language "*components reacts ... by executing instructions of said component on the server.*" Applicant argues that HTML elements, for example, as disclosed in Truong can not be interpreted as components in the sense of claim 3, because there is no suggestion or teaching in the reference that HTML elements contain instructions and execute said instructions on the server.

Claims 4-5 are dependent on Claim 3, and for all the same reasons, applicant submits that claim 4-5 are not taught or suggested by Truong, either alone or in combination with D'Arlach.

With regard to claim 4, the examiner cites D'Arlach (Col. 4:64-65). The cited portion describes that elements have attributes or properties associated with them. The Examiner apparently interprets these attributes and properties as being equivalent to components. This is not possible however because claim 2 requires components to be executed by the page generator and claim 3 requires components to contain instructions. The Examiner has not pointed to any suggestion or teaching that attributes or properties can be executed or contain instructions.

Claim 4 requires one component to contain at least one other component. The Examiner cited prior art where elements contain attributes and properties,. However, since neither attributes nor properties nor elements can be properly interpreted as components, this citation is ineffective to obviate claim 4.

On page 16 of the Final Rejection, the Examiner discusses claims 4 and 5 in the context of claim 3, arguing about HTML elements. These arguments are not applicable to claim 4 and 5 because these claims do not refer to HTML elements.

With regard to claim 5, the examiner cites Figure 5, item 506 of D'Arlach. Step 506 refers to selecting a template for creating or editing a site. The cited portion does not teach or suggest generating different pages for different requests. As explained above, D'Arlach generates pages during publishing, not upon a user request. This means that a page, once

published, stays the same for each page request. In contrast, claim 5 requires a varying set of components for different requests of the same page template.

With regard to claim 64, the examiner cites Truong (Col. 9:15-20) as disclosing components including instructions for use by the document generator to generate browser code. The cited portion refers to javascript code for execution inside the web browser. It does not teach or suggest anything about components. In contrast, applicant's claim explicitly requires instructions for use by the document generator and instructions contained in components.

Claims 65-66 are dependent from Claim 64, and for all the same reasons, applicant submits that Claims 65-66 are likewise not taught or suggested by Truong, either alone or in combination with D'Arlach.

With regard to claim 66, the examiner refers to his reasoning towards claim 2. In addition to applicant's response to the reasoning of claim 2, the cited portions of D'Arlach do not reveal the operation of adding a component to a document template as required by the claim.

With regard to Claim 94, the examiner cited Truong (col. 2:1-10) as disclosing instructions contained in the components. The cited portion discusses various HTML elements, but does not disclose any generation process.

Claims 95 depends from Claim 94, and for all the same reasons are patentable over the cited combination.

4. Claims 6-8

Independent Claim 6 and its related dependent Claims 7-8 should be considered separately and apart from the other claims because these claims introduce the important feature of software components residing on the server that contain executable instructions to react interactively on user input, called interactive software components. In Claim 6, this notion is claimed by reciting a "*at least one selected component that reacts interactively . . . by executing instructions contained in said component on the server.*" Further, Claim 6 requires that one of these selected components be contained somewhere in the data store on a page template "*at least one page template having at least one selected component incorporated therein.*" A component that has this combination of features is not taught or suggested by the cited combination (*see* section A.3). Therefore, applicant submits that the claim is patentable over the cited art.

In the Final Action dated 6/6/05, the Examiner cited Truong (col. 2:1-5) as disclosing a server comprising a plurality of components residing in the data store on the server, including

components that react interactively on user input by executing instructions on the server. However, Truong refers to browser built-in HTML elements, like HTML forms and HTML fields. Since these elements are built into the browser, they reside in the data store of the client and they are executed on the client, not on the server. HTML forms and fields can send data to the server, but they do not contain instructions for execution on the server. This should be clear since the complete browser with everything in it runs on the client only. In contrast, the claim clearly requires that components contain instructions on the server. (See also section A.3)

Further, the Examiner cites D'Arlach (col. 5:25-45) as supporting his interpretation that the recited components are somehow equivalent to D'Arlach's elements. Section A.3 above makes clear that this interpretation is erroneous, because the claim requires components to contain instructions "*by executing instructions contained in said component on the server.*"

In the advisory action, the Examiner mistakenly argues that executing instructions of a component is equivalent to modifying a template and displaying it on the browser. Apparently, the Examiner interprets some part of D'Arlach's editor as being equivalent to a component as recited in applicant's claim. This argument has already been discussed extensively in section A.3 above. Since the claim requires components to be on page templates ("*at least one page template having at least one selected component incorporated therein*"), editor parts cannot be interpreted the same as the recited components.

On page 17 of the Final Rejection dated 6/6/05, the Examiner requests that Applicant identify a feature in Claim 6 that would effectively distinguish Applicant's components from the elements discussed in D'Arlach. Applicant believes that "*executing instructions contained in the component on the server*" is such a distinctive feature. Applicant previously amended claim 6 by adding the language "*executing instructions contained in said component on the server,*" which clearly requires that components contain instructions for execution on the server. The Examiner has never commented on this specific amendment, and appears to maintain his argument based on the prior version of the claim.

In fact, the Examiner does never explicitly argue that D'Arlach's elements contain instructions or are executable. However, on page 16 of the Final Rejection, the Examiner explicitly disagrees with Applicant's assertion that D'Arlach's templates do not have **software** components. The Examiner further states that templates are customizable and contain objects and elements which can be customized. Simply because these elements are customizable, however,

does not teach or suggest that they could contain instructions or be executable. On page 16 of the Final Rejection, the Examiner further argues that storing elements in databases does not matter. However, since databases are commonly used to store data, and not executable instructions, the choice by D'Arlach to use a database for his templates indicates that he sees templates and elements as data, rather than as programs. In fact, the description of the database structure in D'Arlach (*see* col. 5:1-15) demonstrates that the database does not contain instructions for execution on the server nor does it contain executable components. This cited portion describes the two kinds of elements contemplated by D'Arlach: a text element and a button element, both of which do not contain instructions for execution on the server.

The Examiner gives further examples of elements, such as graphical icons, text boxes or buttons (*see* Final Rejection at p. 17), and states that in contrast, java beans are different. The reason that java beans are different, however, is because they contain instructions and are executable, just like applicant's components, while D'Arlach's elements are not.

Claims 7-8 depend from Claim 6, and for all the same reasons are patentable over the cited combination.

With regard to claim 7, the examiner refers to his reasoning in rejecting claim 5. However, claim 5 deals with a varying set of components. In contrast, claim 7 requires "*instructions for interactively editing selected components.*" As pointed out by applicant with respect to claim 2, Truong's editor does not have a specific function for "*interactively editing selected components*" - it just has a function for editing characters of the source code. As explained above, D'Arlach also fails to teach or suggest software components.

With regard to claim 8, the examiner cited D'Arlach (col. 4:60-65) as teaching components inside templates and objects/properties or attributes within a component. However, the cited portion refers to site templates, elements, objects, properties and attributes, but not to components. According to claim 6, components must contain instructions for execution on the server. D'Arlach's templates, elements, objects, properties and attributes do not contain instructions for execution on the server - all of these items are merely data items stored in a data base as explained by D'Arlach at col. 5:15-16. Consequently, none of these items are effective as prior art for components in the sense of base claim 6. In addition, according to claim 6, components are incorporated inside page templates, which means that page templates do not qualify as components. Attributes and properties are not individual objects and therefore can not

be interpreted as components. Since the templates, elements, objects, properties and attributes are not components in the sense of the claim, D'Arlach does not teach or suggest nested components as claimed.

5. Claims 125-127

Independent Claim 125 and related dependent claims 126-127 form a third group considered to stand or fall together, and not with other claims or groups. Claim 125 is the only independent claim directed to a method, and for that reason alone, this claim stands apart from the others. The method is useful for editing an application that is built using components and that operates by generating documents. The method utilizes interactive components, just like many other of applicant's embodiments.

The first step is "running the application, thereby executing selected components." As has been extensively explained in section A.1 and with regard to claim 1, neither of the cited references performs this step during editing – they do not run the application, and they do not teach or suggest components that can be executed (e.g. that contain executable code). For these reasons, this claim is patentable over the cited combination.

The claim also refers to "*executing selected components*" and requires "*selected component in the source code of the application*." For all the reasoning given in section A.3, Applicant submits that the cited prior art combination does not teach or suggest anything like Applicant's components as required by the claim.

Claims 126-127 depend from Claim 125, and for all the same reasons are patentable over the cited combination.

With regard to claim 126, the examiner cited Truong (col. 10:45-50) as disclosing repeating the running and the displaying steps after applying a modification function. However, applicant submits that the cited portion does not teach or suggest the step of running the application. Claim 126 requires that the running step to be repeated. On page 19 of the Final Rejection, with respect to claim 126, the examiner refers to step 116 in Fig. 3A for repeating the running step. In column 9, line 33, step 116 is described. It displays an error message in case the user entered a wrong ID or Password and repeats the login. Applicant submits that the login step in Truong can not be interpreted as the displaying step or the running step. In addition, with a wrong login no modification takes place in Truong.

With regard to claim 127, the examiner cites Truong (col. 8:45-52) for collecting edit information for use by the identifying step. However, Truong teaches identifying files for editing. In contrast, the identifying step of claim 125 refers to identifying a component on a page template. On page 19 of the Final Rejection, with respect to claim 127, the examiner asserts that identifying from Truong is identical to Applicant's identifying as claimed. According to Truong at col. 8:45-52, Truong's identifying step identifies files. According to the claim language in claim 125 "*identifying the selected component*," applicant's identifying step as claimed refers to components, not files. Therefore, Truong's identifying step can not be correlated with applicant's identifying step.

6. Claims 90-93 and 96

Independent Claim 90 is directed to an editor embodiment. The editor is used with a web browser and allows a user to edit a document being actively displayed by the browser "wherein scripts contained in said document remain functional during editing." Further, the editor includes a client part, e.g., a first software program for execution within the browser that initiates editing functions when the user clicks on the displayed document. Because of these specific limitations, claims 90-96 should be considered as a group separately from the other groups.

The Examiner states that Truong discloses an editor that allows the user to edit a document being displayed by the browser, citing col. 1:65-67, col. 2:1-10, and col. 2:17-30. The cited portions do not reveal any substantive details about Truong's editor, but instead provide background regarding web technology, such as form capable browsers. Such browsers can edit text contained inside form fields displayed on an HTML page. However, this is much different than editing the HTML page itself. In fact, Truong's editor is made for editing a file displayed inside a form field and not for editing the displayed HTML page itself. (See Truong at Fig. 5 and col. 10:45-50).

The Examiner also asserts that Truong discloses scripts that stay functional during editing, citing col. 7:1-5. However, the cited portion of Truong reveals only that the browser is capable of processing scripts. Such disclosure does not teach or suggest that the file being edited is sent to the browser in such a way that the browser recognizes and executes the scripts during editing. In fact, to the contrary, Truong specifically discloses that the text of the file is sent to the browser in such a way that it can be edited *as text*, see col. 10:45-50, which implicitly means that the scripts are not executed by the browser, but are instead shown in source code form. In

addition, Fig. 5 of Truong shows the editing of a script, which further supports applicant's contention that according to Truong, scripts in the selected file are shown as text and are not executed.

In applicant's editor, as recited in Claim 90, "*scripts contained in said document remain functional during editing.*" In Truong's editor, they do not. The Examiner suggests that "*sending text back to the browser*" as disclosed by Truong is the same thing, but in fact, such a step creates a new document in the browser – it is not the same document. The claim, however, specifically requires that scripts in "*said document*" remain functional during editing.

The Examiner also cites Truong (col. 8:20–25), but the cited portion shows a script, which is part of the editor used to check login ID and password. This shows a single script, which is executed before editing and not during editing. Claim 90, however, requires that "*scripts contained in said document remain functional*" Fig. 5 of Truong clearly shows editing of a script which is not executed during editing. This shows that not all scripts contained in said document remain functional.

The Examiner cites Truong (col. 10:45–50) as disclosing *the editor including a first software program* for execution within the browser for processing selected clicks on the view of said document displayed by the browser. The cited portion reveals that "the text may be edited at the web browser using editing features." However, this refers to the browser built-in editing functions of an HTML text area, as shown Truong's Fig. 5. The cited portion, therefore, seems to reveal program instructions inside the browser only. In contrast, Claim 90 requires a first software program included in the editor.

Figure 2 shows that D'Arlach's editor is a software program for running on the server only. Figure 2 does not show the editor including a first software program for execution within the browser on the client. In contrast, significant parts of applicant's editor are implemented as scripts for execution within the browser. Advantageously, this works much faster because fewer server interactions are needed. D'Arlach describes (see col. 6:3-5) that the editor immediately requests the server when the user clicks on a selected portion. In contrast, applicant's claim explicitly requires a first software program for processing selected clicks.

Truong uses browser built-in text boxes for editing, including associated program instructions. As already discussed, these instructions are part of the browser only and not part of Truong's editor. In addition, these instructions can not be used and combined with D'Arlach

because scripts contained in HTML text boxes are displayed as source code and are not functional as required by the claim. Applicant therefore submits that because neither D'Arlach nor Truong has a first software program as claimed, Claim 90 is patentable over the combination of Truong and D'Arlach.

Claims 91-96 depend from Claim 90, and for all the same reasons are patentable over the cited combination.

With regard to claim 91, the examiner cited Truong (col. 6:55-65) as disclosing the editor using a second browser window for showing information on elements contained in a document. However, the cited portion discloses HTML elements, but does not talk about browser windows. In addition, there are no details provided on the user interface of Truong's editor, merely information on the web browser used. On page 18 of the Final Rejection, with regard to claim 91, the examiner argues that Truong discloses editing in two windows. He refers to Fig. 3A, step 102, and step 108. According to col. 9:1-10, step 102 involves starting the browser, and according to col. 9:20-22, step 108 means displaying the network editor web page inside the browser. Thus, step 102 starts the browser in a single window and step 108 loads the network editor web page into it. This does not indicate that the browser shows two windows, but shows the same browser window at different times. Therefore, applicant submits that the examiner is wrong in interpreting Truong to use two browser windows. Truong just uses a single browser window during editing. In contrast, claim 91 explicitly requires a "second browser window".

With regard to Claim 92, the examiner cited Truong (col. 10:45-50) as disclosing storing modification on said document in cooperation with the first software program. However, the cited portion reveals transmitting the content of a file to the web browser. This is clearly different from modifying a document in cooperation with the first software program.

Claims 93-95 depend from Claim 92, and for all the same reasons are patentable over the cited combination.

With regard to Claim 93, the examiner cited Truong (col. 7:20-30) as disclosing that a generated document looks similar to the original. The cited portion talks about generating a document for display to the user, but does not indicate that the generated document looks similar to any other document. On the contrary, Truong's Figure 5 shows clearly that the editor displays only the source code text, which looks, in the case of HTML documents, very different than the original view of an HTML document.

With regard to claim 96, the examiner cites Truong (col. 8:39-45) as disclosing that links contained in the document stay functional during editing. However, the cited portion refers to the file selection form, not to the document being edited, and not to links. Truong's editor shows links during editing as source text, and not as functional links, as required by the claim. Also, D'Arlach's editor does not show functional links. As described in col. 9:13-14 of D'Arlach, clicking on the "Phone Services" button during editing does not operate the link as in the user's view, but instead, brings up the "Elements Properties" screen. This makes browsing using the normal link navigation of a web site impossible during editing. In contrast, applicant's claim requires that links stay functional during editing in order to allow the user to browse and edit at the same time.

7. Dependent Claims 28 and 29

Claim 28 is dependent from claim 27, and adds the limitations that the second document includes handles, and that choosing a handle selects a component for editing. Claim 29 depends from claim 28 and adds the limitation that the handle indicates the position of a component on the first document. Because of these limitations, claims 28 and 29 should be considered separately.

With regard to claim 28, the examiner cites D'Arlach (col. 3:33-45) and refers to selecting and editing templates which include editable components and objects. However, the cited portion is an introductory remark that does not refer to page templates or selecting. In particular, the cited portion does not specify how an element is selected, nor does it mention anything like a handle. D'Arlach does not disclose the use of handles, but instead, elements are selected by directly clicking on them. As described in col. 9:13-14 of D'Arlach, clicking on the "Phone Services" button itself, and not on a handle, brings up the "Elements Properties" screen. In addition, Fig. 10 does not show anything like a handle. In contrast, applicant's claim explicitly requires handles.

On page 17 of the Final Rejection, in response to applicant's arguments regarding claims 28 and 29, the examiner states that handles are inherent in graphical fill in textboxes, which are used to increase size on all corners of the box with a mouse. Applicant disagrees such handles are not inherent to graphical fill in textboxes created with HTML. HTML fill in textboxes displayed by a browser do not have handles and do not provide a function to increase the size with the mouse. The examiner apparently suggests with this interpretation of handles that

D'Arlach's editor provides handles, simply because HTML textboxes do. However, the HTML textboxes and all the other classical HTML elements used by D'Arlach or Truong do not have handles and do not provide resizing functionality. Applicant therefore submits that neither D'Arlach or Truong show a prior art editor using handles.

Claim 29 depends from Claim 28, and for all the same reasons are patentable over the cited combination.

8. Dependent Claims 41 and 42.

Claim 41 depends upon claim 1, and claim 42 depends from claim 41. Claim 41 and the notion of a client part while claim 42 requires the client part to include instructions for execution that are automatically downloaded from the server. Because of these limitations, claims 41 and 42 should be considered separately.

With respect to claims 41 and 42, the examiner cites Truong (Col. 3:30-38) and specifically refers to the editor input form and the editor selection form. However, the editor selection form is used by Truong to select a file for editing. The cited portion does not teach or suggest that the editor selection form is an executable program. In contrast, claim 41 requires a client part for execution on the client computer. The cited portion does not show any executable instructions. In contrast, claim 42 explicitly requires instructions. Applicant's editor has a client part, consisting of many instructions, that is downloaded to the client computer when the editor is started. On page 16 of the Final Rejection, the Examiner cites Truong (Col. 6:32-35) for its teaching that "Processor 34, interprets and executes instructions that have been fetched or retrieved from client library." However, the Examiner has misquoted Truong, and the cited portion actually reads "Processor 34, interprets and executes instructions that have been fetched or retrieved from client memory." Thus, the cited portion talks about the general workings of the client computer and does not reveal anything about the operation of the editor itself.

C. CLAIMS 22-24, 26, 27, 30-33, 43, AND 67 ARE PATENTABLE UNDER SECTION 102(E) OVER TRUONG

1. Claims 26, 27, 30-33, and 43

Independent Claim 26 and related dependent Claims 27-33⁵ and 43 stand or fall together as a fourth grouping of Claims. Claim 26 does not refer to "components" like most of the other groups. Further, Claim 26 uses the language "*to modify documents*" while most of the other claims refer to editing "*document templates*." Thus, this group of claims merits separate consideration.

Claim 26 describes a system to modify a first document stored on a server whereby a second document is displayed to the user that appears and functions similar to the first document. Applicant emphasizes the fact the recited elements of the claim are located on the server, wherein the server comprises:

- a document store;

- a first software program including instructions for transforming at least one first document into a second document having features which permit editing of the first document such that at least a part of the second document appears and functions the same as the first document; and

- "a second software program including . . . instructions to modify the first document,*

Claim 26 stands rejected on the basis of anticipation by Truong. However, Truong clearly does not teach or suggest the combination of elements recited above. Truong merely discloses a web based editor for plain text files. When applied to programs or HTML pages, it just displays the source code without following the WYSIWYG principle. Truong thoroughly discusses functions for editing plain text in his specification. (*See*, for example, col. 3:40-42, col. 10:40-50, and Fig. 5). Since plain text is not functional, and Truong just displays the plain text, Truong's editor does not operate in a way that "*at least a part of the second document appears and functions similar to the first document.*" However, this is exactly what is required by the claim.

Nevertheless, the examiner rejected Claim 26 on the basis that Truong (col. 3:27-38) discloses a first software program including instructions for transforming a first document into a second document having features which permit editing of the first document such that at least part of the second document appears and functions similar to the first document. However, the Examiner is in error. The cited portion of Truong discloses that an editor selection form is displayed which lists filenames for the user to select for editing. The examiner explicitly mentions the editor selection form (*see* Final rejection at p. 3), apparently as the second document. Since the editor selection form lists the filenames only, but not the file content, it can

⁵ Claims 28 and 29 were rejected on a different basis, but because applicant asserts that the rejection of base Claim 26 was in error, these Claims are considered in this grouping.

not possibly appear or function similar to the document being edited. In contrast, applicant's Claim explicitly requires that the second document appear and function similar to the first document.

On page 15 of the Final Rejection, in responding to applicant's arguments regarding Claim 26, the Examiner mischaracterizes applicant's argument by stating that "Applicant argues that Truong only edits file list and not the file content." In fact, applicant stated that "*Truong explains the display of an editor selection form which lists filenames for the user to select for editing.*" Certainly Truong's editor ultimately edits the content of the file selected. However, Claim 26 requires this to be done in a specific way, i.e., by transforming a first document into a second document such that the second document appears and functions similar to the first document. The portions of Truong cited by the examiner (col. 3:27-38; Fig. 4) do not give any details on how the editing of the file content takes place, because these portions only discuss listing the file names. On the other hand, Fig. 5 of Truong does show the view of the editor while editing the file content, and it shows the source code of the file which is clearly not functional and looks different from the normal view.

Thus, Truong merely discloses a web-based text editor which does not permit editing of fully functional, running web applications, and does not teach or suggest, explicitly or inherently, the use of a first software program to transform a first document into a second document that appears and functions similar to the first document, and a second software program having instruction for modifying the first document. Therefore, Claim 26 is patentable over Truong.

Claims 27-33 and 43 depend from Claim 26, and for all the same reasons are patentable over the cited combination.

Applicant submits that claim 27 is patentable over Truong and D'Arlach for similar reasons as claim 51, i.e. components. Applicant therefore would put claim 27 into the same group as claim 51. The examiner rejected claim 27 based on Truong alone, not in combination with D'Arlach, so claim 27 is discussed in this section C rather than in section B.

With regard to dependent claim 27, the examiner cites Truong (col. 8:13-15) as disclosing a component being executed by the first software program. The cited portion refers to the editor input form, which is a browser built-in HTML form. In contrast, applicant's claim requires that the component be executed by the first software program that does the document translation. Truong makes no such teaching or suggestion.

On page 15 of the Final Rejection, with regard to claims 27, 30-33 and 43, the examiner states “*Applicant argues that Truong does not disclose the editor input form, which is a browser built in HTML.*” This is certainly not a correct characterization of applicant's argument. Applicant stated with regard to claim 27 that “*the examiner cites Truong at column 8 lines 13-15 as disclosing a component being executed by the first software program. The cited portion refers to the editor input form, which is a browser built-in HTML form.*” Applicant never argued, as the examiner states, that Truong does not disclose an editor input form, but on the contrary states that Truong column 8 lines 13-15 disclose an editor input form.

Applicant's claim, however, requires a component being executed by the first software program that does the document translation. The examiner did not cite prior art for this kind of component. The examiner did cite a text portion discussing the editor input form. The editor input form, however, is part of the browser on the client and therefore not executed by the first software program that runs on the server. Thus, the editor input form can not be interpreted as being the same as a component in the context of claim 27.

With regard to dependent claim 30, the examiner cites Truong (col. 7:1-5). However, the cited portion appears to simply describe that a browser can execute scripts. In contrast, applicant's claim explicitly requires that scripts may be used as “*features which permit editing of the first document such that at least part of the second document appears and functions similar to the first document . . .*” Truong fails to teach or suggest that scripts could be used in this way.

Claim 31 depends from Claim 30, and for all the same reasons are patentable over the cited combination. With regard to dependent claim 31, the examiner cites Truong (col. 9:15-20). However, the cited portion discusses using scripts to do local processing. In contrast, applicant's claim specifically requires that the scripts encapsulate information from the first document. There is no such teaching or suggestion in Truong.

With regard to dependent claim 32, the examiner cites Truong (col. 9:10-15). However, the cited portion refers to an editor input form which shows only the source text. In contrast, applicant's base claim 26 requires “*features which permit editing of the first document such that at least part of the second document appears and functions similar to the first document*” and claim 32 adds “*wherein features incorporate information regarding the first document into the second document.*” The examiner did not cite prior art for such features. The editor input form can not be interpreted as such a feature because the source text of a web page displayed in the

editor input form does not function and certainly looks drastically different from its normal view. Thus, Truong does not teach or suggest such features.

Claim 33 depends from Claim 32, and for all the same reasons are patentable over the cited combination. With regard to dependent claim 33, the examiner cites Truong (col. 7:10-30). However, the cited portion explains the general operation mechanism of a web browser. It does not specifically refer to change requests. In contrast, claim 33 explicitly requires the information incorporated into the second document be used "*to send change requests for the first document to the server.*" Truong does not teach or suggest this feature. Instead, Truong sends the complete file from the client to the server and saves it under the filename. (See Truong at col. 10:55-59). This shows that Truong does not send change requests, but the complete file. In contrast, claim 33 requires "change requests."

With regard to dependent claim 43, the examiner cites Truong (Col. 3:30-40 and Col. 9: 18). The first cited portion refers to various actions without specifying if they are done by a script or by the normal browser function controlled by HTML. The second cited portion refers to scripts contained in the second document. In contrast, applicant's claim 43 requires "*at least one additional script that works in cooperation with the second document.*" There is no such teaching or suggestion in Truong.

On page 16 of the Final Rejection, the examiner states "with regards to HTML Applicant again is arguing for an unclaimed merit of distinction." This is not a correct characterization of applicant's argument. The examiner cited Truong at Col. 3:30-40. The cited portion discloses the action of sending a request. This can typically be done with either HTML or with a script. The cited portion does not disclose the use of scripts. In contrast, applicant's claim requires "at least one additional script."

2. Claims 22-25

Independent Claim 22 and related dependent Claims 23-24 stand or fall together as a fifth grouping of Claims by virtue of their rejection under Section 102(e). This group of Claims is directed to a computer that includes a document generator executing components, and an editor editing these components. This claim has similarities to claim 51, and the arguments for claim 51 and section A.3. have relevance here as well.

Claim 22 stands rejected as anticipated by Truong. The examiner cited Truong (col. 11:17-20) as disclosing an editor operable within the web browser for inserting, deleting, and

modifying components on document templates. However, the Examiner is in error. The cited portion discloses that common WINDOWS editing commands, such as copy, insert, delete, and paste, may be used if the browser is running on the WINDOWS operating system. The cited portion does not teach or suggest that the editing commands would interact with "components" on document templates. In contrast, at column 10 line 47, Truong writes "the text may be edited at the browser using editing features such as delete, select, search, copy, paste, and the like." This makes it clear that Truong's editing features work only on text, and not on components.

The examiner also cited Truong (col. 10:45-50) as disclosing a document generator for processing document templates, executing components and for generating documents from the document templates that are understandable by the web browser. However, the Examiner is again in error. The cited portion discloses providing the text of the selected file to the web browser. It does not teach or suggest the use of components, or a document generator for executing components, as required by the Claim.

Claims 23-25⁶ depend from Claim 22, and for all the same reasons, Claims 23-25 are not taught or suggested by Truong, either alone or in combination with D'Arlach.

With regard to Claim 23, the examiner cited Truong (col. 2:1-5; col. 10:45-50) as disclosing that the editor operates functional applications in an edit mode thereby permitting editing directly in the web browser. However, the Examiner is in error. The cited portion actually discloses the editing of source text in the web browser, and there is no teaching or suggestion that functional applications are running in an edit mode. Truong actually teaches away from such a suggestion by stating the editor edits the plain text (*see* col. 10:45-50), and Fig. 5 shows an example. In contrast, applicant's editor operates a functional application in an edit mode thereby permitting editing of generated pages directly in the browser.

Claim 24-25 depend from Claim 23, and for all the same reasons are patentable over the cited combination. With regard to Claim 24, the examiner cited Truong (col. 10:45-50) as disclosing a component that can react on subsequent document requests by executing selected instructions. The Examiner is in error. The cited portion actually describes instructions of part of the editor. According to the base claim 22, the editor is specifically for inserting, deleting, and modifying components on document templates. Truong's editor is not, however, specifically for

⁶ Claim 25 was rejected on a different basis, but may properly be considered with this group.

inserting, deleting, and modifying such editor parts. In contrast, Claim 24 requires that the instructions belong to the component and be executed upon subsequent document requests.

Claim 25 depends from Claim 24, and for all the same reasons is patentable over the cited combination.

3. Dependent Claim 67

Claim 67 depends from Claim 59, and applicant believes that Claim 67 can properly be considered together that claim.

Applicant notes that Claim 67 is dependent from Claim 59, which was rejected based on the combination of Truong and D'Arlach. Thus, it is inconsistent and improper to reject this claim as anticipated by Truong alone. However, for all the same reasons that Claim 59 is considered patentable, these Claims should also be considered patentable.

With regard to Claim 67, the examiner cited Truong (col. 8:39-50) as disclosing that the view looks similar to the end-user view of the generated document, except for editing features. However, the Examiner is in error. The cited portion of Truong discloses the file selection form, but does not give any indication that the view is similar. On the other hand, Fig. 5 clearly indicates that Truong's editor shows the source code of any document, and it is well known that the source code of an HTML document does not look similar to its end-user view.

D. CLAIM 25 IS PATENTABLE UNDER SECTION 103(A) OVER THE COMBINATION OF TRUONG, D'ARLACH AND U.S. PATENT NO. 6,651,108 TO POPP ET AL.

Claim 25 depends from Claim 24, and for all the same reasons, applicant submits that Applicant believes that Claim 25 can properly be considered together with the claims identified in D.2. above.

Claim 25 is not taught or suggested by Truong in combination with D'Arlach and Popp. For these reasons applicant would put Claim 25 into the second group of claims with Claim 51 and Claim 22.

The examiner acknowledges that neither Truong nor D'Arlach discloses component classes implementing components, (See Final Rejection on page 14). The Examiner thus uses Popp to support the notion that the use of classes is a well known practice in an object oriented environment. However, neither Truong nor D'Arlach uses an object oriented environment. As

previously discussed, Truong's browser built-ins are not executed by the document generator, as required by the base claim, Claim 22. Further, D'Arlach's elements are data items stored in a database, and are not capable of execution, as explained above. Thus, there would be no need or motivation to implement D'Arlach's elements by any technology. Thus, the addition of Popp adds nothing and is therefore ineffective and improper.

VIII. CLAIMS APPENDIX

1. A software development system for applications that run on a data network which couples a server computer and a client computer, wherein the client computer runs a browser program, comprising

a page generator running an application being developed and sending generated documents to the browser for display as pages including additional editing features for interpretation by the browser program;

an editor directly operating on the pages displayed by the browser via the editing features, thereby allowing the user to work on a functional application during development.

2. A software development system as claimed in claim 1, further comprising a plurality of components, and wherein developed applications comprise at least one page template capable of containing components, and wherein the editor provides features to insert, modify and delete a component on at least one page templates, and wherein the page generator executes selected components on page templates.

3. A software development system as claimed in claim 2, wherein at least one of the components reacts interactively on user input by executing instructions of said component on the server.

4. A software development system as in claim 3, wherein at least one of the components contains at least one other component.

5. A software development system as in claim 3, wherein the set of components on pages generated from at least one page template can vary for different requests of said page template.

6. A software development system for use in a data network which couples a server computer to a client computer, wherein the client computer includes a first software program for generating a request for one or more pages from the server computer and for displaying pages, and wherein the server computer includes a second software program for receiving and processing the request from the client computer, for generating and storing pages, and for

transmitting pages to the client computer in response to requests, the server computer further comprising:

a data store,

a plurality of components residing in the data store, including at least one component that reacts interactively on user input by executing instructions contained in said component on the server;

a plurality of page templates residing in the data store, at least one page template having at least one selected component incorporated therein; and

a server processor controlled by a third software program, said program providing instructions for selecting a page template based on the request from the client computer and instructions for generating a page from the page template for transmission to the client computer.

7. The development system of claim 6, further comprising a component editor controlled by a fourth software program, said program providing instructions for interactively editing selected components on a page template.

8. The development system of claim 6, wherein a component is nested within a component.

9. A method for generating documents for display by a browser using components that react interactively on user input by executing instructions on a server, comprising the following steps for execution on the server upon a document request:

assigning a unique identifier to at least one of the components; and
embedding the unique identifier into a generated document.

10. The method of claim 9, further comprising storing data on the server representing at least one of the components.

11. The method of claim 10, further comprising:
analyzing the request sent by the browser for unique identifiers; and

calling a function for the interactive components referenced by at least one of the unique identifiers contained in the request.

12. The method of claim 11, wherein at least one of the components is contained on a document template.

13. The method of claim 11, wherein at least one of the components is called by a program.

14. The method of claim 11, wherein at least one of the components is called by another component.

15. The method of claim 11, wherein the data is stored into an object of an object oriented programming language and wherein the function is a method of the object.

16. A method for implementing client server applications, comprising:
storing data objects on a server and assigning a unique identifier to each data object;
dynamically generating a document with the unique identifier embedded in the document;
and
analyzing requests for unique identifiers and calling at least one function for a data object associated with one of the unique identifiers found in the request.

17. The method of claim 16, wherein the unique identifier is embedded inside a uniform resource locator contained in a tag of the document.

18. The method of claim 16, wherein the unique identifier is embedded in scripts contained in the document.

19. The method of claim 16, wherein the unique identifier is unique within a single session.

20. The method of claim 16, wherein the unique identifier is unique within all documents generated by a single server within a defined time period.

21. The method of claim 16, wherein the data objects are created by an object-oriented programming language and said function is a method of one of these objects.

22. A computer running an application to develop and maintain applications using a web browser, comprising:

an editor operable within the web browser for inserting, deleting, and modifying components on document templates; and

a document generator for processing document templates, executing components and for generating documents from the document templates that are understandable by the web browser.

23. A computer as in claim 22, wherein the editor operates functional applications in an edit mode permitting editing directly in the web browser.

24. A computer as in claim 23 wherein at least one of the components contains instructions and can react on subsequent document requests containing user responses by executing selected instructions of said component.

25. A computer as in claim 24, wherein the computer further comprises:
a store of component classes, each component class implementing one component kind;
and

a parser able to detect components marked on document templates;
wherein the document generator works upon a document request using component classes to generate browser code; and

wherein the editor is capable of showing a menu of components for insertion into the document templates.

26. A system to modify documents on a server in a data network which couples said server computer to a client computer, the server computer comprising:

a document store;

a first software program including instructions for transforming at least one first document retrieved from the document store into a second document having features which permit editing of the first document such that at least a part of the second document appears and functions similar to the first document; and

a second software program including instructions to receive information from the client computer and instructions to modify the first documents stored in the document store.

27. The system of claim 26, wherein the first document includes at least one component being executed by the first software program.

28. The system of claim 27, wherein the second document includes handles and choosing one of the handles selects a component for an editing operation.

29. The system of claim 28, wherein at least one handle indicates the position of at least one component contained in the first document and said editing operation includes modifying the component, deleting the component, and displaying information regarding the component.

30. The system of claim 26, wherein the features include scripts.

31. The system of claim 30, wherein the scripts encapsulate information from the first document.

32. The system as in claim 26, wherein the features incorporate information regarding the first document into the second document.

33. A system as in claim 32, wherein the information incorporated into the second document is used on the client computer in order to send change requests for the first document to the server.

34. A method for generating a document for display in a browser from a document template containing components, comprising:

for each component denoted on the document template, identifying a component class of the component; and

based on data contained in a request initiated by the browser storing a first object of the component class, the first object representing the component.

35. The method of claim 44, further comprising calling a method of said component class to generate browser code, said method being the constructor.

36. The method of claim 34, further comprising, for all components having a name attribute, looking up the component object in session memory based on said name attribute.

37. The method of claim 34, further comprising, for at least one component kind, for all components denoted on the document template having said component kind;

generating a unique identifier;

assigning said unique identifier to said object, and

embedding said unique identifier into the browser code.

38. The method of claim 37, further comprising:

inserting objects for the components of at least said component kind into a list of listening components;

working through all objects stored in the list of listening components whose unique identifier occurs inside a name in the form data set; and

calling a method of at least one of these objects.

39. The method of claim 34, wherein the document template is parsed into a list of nodes, including text and component nodes, said method further comprising:

determining if the current node is text or a component;
if component, then calling a method for the component, comprising:
 evaluating the attributes of the component if necessary;
 identifying the component class associated with the component; and
 calling the constructor method of the component class,
 said constructor method generating browser code;
if text, then generating the text; and
repeating these steps for each node.

40. The method of claim 39, wherein at least one component contains nested components and the method of claim 39 is recursively performed for all nodes nested inside the component.

41. A software development system as in claim 1, the editor comprising a client part for execution on the client computer.

42. A software development system as in claim 41, wherein the client part comprises instructions that are automatically downloaded from the server prior to editing.

43. A system as in claim 26, additionally comprising at least one script for automatic download to the client that works in cooperation with the second document to permit editing of the first document.

44. The method of claim 34 wherein storing the first object comprises creating a new object as necessary.

45. The method as in claim 34 wherein components are denoted on document templates using tag syntax.

46. The method as in claim 45 wherein the tag name identifies a component class.

47. The method as in claim 36 wherein components are denoted on document templates using tag syntax, wherein the tag name identifies a component class.

48. The method as in claim 36 wherein the component object, if found, is reused to store the first object.

49. The method as in claim 36 wherein in case a component has a name attribute but no component object is found a new object is created and stored under said name in session memory.

50. The method as in claim 49 wherein new objects are created for all components not having a name attribute.

51. A system for editing components on web document templates for use with a first software program including first instructions for generating a document request to obtain at least one generated document from a second software program and for displaying the generated document, the second software program capable of receiving and processing the document request and of transmitting first documents to the first software program in response to requests, said system comprising:

a plurality of components containing instructions to generate browser code,

a plurality of document templates,

the second software program transmitting, while processing selected requests, second documents to the first software program that make the first software program display a user interface for editing functions used for maintaining components on document templates,

a third software program used by the second software program while processing selected document requests, the third software program including third instructions for modifying document templates in order to perform said editing functions.

52. The system of claim 51, wherein at least some components include fourth program instructions including steps to generate browser code for transmission to the first software program in response to a request from the first software program, wherein the browser code can differ for multiple requests for the same document template.

53. A system in claim 52 running on a data network, coupling a server computer and a client computer, the first program running on the client computer, the second program running on the server computer.

54. A system in claim 52 wherein second documents include HTML pages with embedded scripts.

55. The system of claim 52, wherein the edit function includes adding a component to a document template, removing a component from a document template, and modifying component on a document template.

56. The system of claim 52, further comprising a fifth software program used by the second software program while processing selected document requests, the fifth software program including fifth instructions for generating generated documents from document templates thereby calling fourth program instructions.

57. The system of claim 56, wherein the generated document includes, if requested in edit mode, edit features for interpretation by the first software program.

58. The system of claim 56 further comprising instructions to allow the user to click on the generated document to select items to perform edit functions on.

59. A software development system for developing dynamic web documents, said dynamic web documents operating by being transformed into an end user's view upon a request by a web browser, the end user's view being provided to the browser in response to the request, comprising:

an editor program for editing dynamic web documents,
a document generator for generating generated documents from dynamic web documents which look and function similar to the end user's view of the documents,
the editor program comprising first instructions for requesting the document generator to process a dynamic web document leading to a generated document,
the system further comprising second instructions for displaying at least some information items contained on said generated document in a view which allows the user to select an item to which a modification function will be applied,
the editor program further comprising third instructions to modify the dynamic web document to perform said modification function.

60. The software development system of Claim 59 running on a data network, which couples a server computer and a client computer, the document generator running on the server computer the editor at least partly running on the client computer.

61. The software development system of claim 60 further comprising fourth instructions for execution during document generation to collect edit-information for use by the editor.

62. The software development system of claim 60, wherein the editor uses a web browser for displaying said view.

63. The software development system of claim 60, able to automatically repeat requesting the document generator to process the dynamic web document if required.

64. The software development system of Claim 59 further comprising a plurality of components including at least one component marked on said dynamic web document and including instructions for use by the document generator to generate browser code.

65. The software development system of claim 64, wherein the editor uses a web browser for displaying said view.

66. The software development system of claim 64, wherein modification functions include insert of a component, delete of a component, and modify a component.

67. The software development system of claim 59, wherein said view looks, except for editing features, similar to the end-user view of the generated document.

68. The software development system of claim 59 further comprising sixth instructions to collect edit-information for use by the editor, said sixth instructions for execution during document generation.

69. The software development system of claim 68, wherein the editor uses the edit-information to correctly modify the dynamic web document.

70. The software development system of claim 69, further comprising a plurality of components wherein the edit-information comprises position information on selected components marked on the dynamic web document.

71. The software development system of claim 59, wherein the editor uses a web browser for displaying said view.

72. The software development system of claim 71, wherein first instructions comprise seventh instructions for initiating a reload in the browser.

73. The software development system of claim 59 wherein the editor program further comprises eighth instructions to display information on at least one element of at least one dynamic web document, that is replaced during document generation, without requesting the document generator to generate a document.

74. A software development system for document templates that are intended for transformation into generated documents for display by a first software program, the first

software program including first instructions for generating a document request to obtain at least one generated document and for displaying the generated document, comprising:

- a plurality of components, that include instructions to generate browser code for transmission to the first software program,
- an editor capable of performing edit functions maintaining components on document templates, the components capable to cooperate with the editor,
- a plurality of document templates having components denoted thereon, and
- a document generator comprising second instructions to, upon document requests, generate generated documents from at least one document template for display by the first software program wherein the set of components on the generated document can vary for different document requests for said document template.

75. The software development system as in claim 74, wherein edit function comprises adding a component, modifying a component, and deleting a component.

76. The software development system as in claim 74, wherein tag syntax is used to denote at least one components on at least one document templates, whereby the tag name identifies the component kind.

77. The software development system of claim 74 running on a data network, which couples a server computer and a client computer, the document generator running on the server computer the editor running, at least partly, on the client computer.

78. The software development system as in claim 74, wherein at least one component, that can upon select react interactively on subsequent document requests, can upon selected document requests for said document template be excluded from said generated document.

79. The software development system as in claim 78 further comprising third instructions to prevent excluded components from reacting on subsequent document requests.

80. A software development system as in claim 79, said third instructions comprising fourth instructions to, upon a first document request, store information in session memory on some of the components, that are present on the document generated based on the first request, and fifth instructions to, upon subsequent document requests, only react on components that have been remembered in session memory thereby avoiding tampering with excluded components on the side of the first program.

81. A software development system as in claim 74 wherein at least one first component contains sixth instructions to decide upon a request for said document template about exclusion of components nested inside the first component from the generated document.

82. A software development system as in claim 74 the system able to provide an editable view taking the varying set of components into account.

83. A software development system as in claim 74 the system able to provide an editable view that includes and excludes selected components on different requests for said document template similarly as the end user's view of the document template.

84. A software development system as in claim 74 wherein a document generated for at least one document template contains more components than the document template for at least one document request.

85. The software development system as in claim 74, wherein multiple instances of at least one third component denoted on the document template can be included in at least one of one of the documents generated from said document template.

86. The software development system as in claim 74, further comprising seventh instructions to assign a unique identifier to each component instance of at least one seventh component, whereby the seventh component includes eighth instructions to qualify names generated into the browser code with the unique identifier.

87. A software development system as in claim 74, wherein-at least one fourth component contains ninth instructions to decide upon a request about how many instances of components nested inside the fourth component are included in the documents generated from said document template.

88. A software development system as in claim 74 the editor able to provide an editable view that include multiple instances of selected component similarly as the end user's view of the document template.

89. A software development system as in claim 74 wherein at least one sixth component includes tenth instructions to display the sixth component, the tenth instructions being used to generate browser code for displaying the sixth component during editing as well as during normal use of the component.

90. An editor for use with a web browser, the editor allowing the user to edit at least one document displayed by the browser, wherein scripts contained in said document remain functional during editing, the editor including a first software program for execution within the browser and for processing selected clicks on the view of said document displayed in the browser by initiating editing functions .

91. The editor as in claim 90 using at least two windows, a first browser window displaying said document and a second window for displaying information on an element contained in said document.

92. The editor in claim 90 further comprising a second software program for storing modifications on said documents in cooperation with the first software program.

93. The editor as in claim 92 further comprising a third program for transforming an original document into the document, the browser displaying the document as said view looking similar to the original document and interpreting editing features contained in the document.

94. The editor as in claim 93 wherein said original document is a dynamic document having components denoted thereon, the third software program further comprising instructions for generating browser code in cooperation with selected instructions contained in the components.

95. The editor as in claim 94 wherein the browser together with the first software program is running on a client computer connected to a server computer via a data network, wherein the second and the third software program run on the server computer.

96. The editor in claim 90 wherein links contained in said document stay functional allowing the user to browse and edit at the same time.

97. A system for displaying dynamically generated documents in a data network coupling a server computer to a client computer, wherein the client computer has a first software program including first instructions for generating a document request to obtain at least one generated document from the server computer and for displaying the generated document, comprising:

a plurality of components for execution on the server computer, including a first component including second program instructions to generate browser code and third program instructions for execution on the server which are initiated by the user interacting with the first component, and,

fourth program instructions on the server computer for, based on data contained in a document request initiated by the first software program on the client computer, generating generated documents for transfer to the client computer and display by the first software program, thereby calling second program instructions of components.

98. The system of claim 97, the server computer further comprising fifth program instructions for analyzing said data and for calling third program instructions of first component as necessary.

99. The system of claim 98, further comprising a plurality of document templates residing in the data store, at least one of the document templates having at least one second component denoted thereon.

100. The system of claim 99, wherein tag syntax is used to denote the second component, whereby the tag name identifies a component.

101. The system of claim 99, wherein at least one third component denoted on a document template contains the first component.

102. The system of claim 101, wherein third components implementation scheme includes logic to decide how often to insert the first component into the generated document.

103. The system of claim 98, the server computer further comprising sixth program instructions for, based on the document request, deciding to insert more than one instance of the first component into the generated document.

104. The system of claim 103, making sure that multiple instances of the first component do not interfere by qualifying names generated into the browser code using unique identifiers.

105. The system of claim 98, the server computer further comprising seventh program instructions for, based on the data, deciding to exclude the first component from the generated document.

106. The system of claim 105, wherein fifth instructions call third instructions only if the first component was contained on a page previously transferred to the client.

107. The system of claim 98, wherein fifth program instructions include eighth program instructions to analyze said data for user interactions with multiple components and to call third program instructions of multiple components as necessary.

108. The system of claim 107, wherein components include ninth instructions to check for errors and fifth instructions include tenth instructions to call ninth instructions of components as necessary and to suppress subsequent calling of third instructions in case of errors.

109. The system of claim 98, further comprising eleventh program instructions for storing a data object in session memory representing at least one component instance included in a generated document, said eleventh program instructions for execution while dynamically generating a document.

110. The system of claim 109, wherein third program instructions are encapsulated in a method of data objects, fifth program instructions including twelfth program instructions for identifying the data object that represents a component instance the user interacted with and for calling said method of said data object as necessary.

111. The system of claim 110, further including thirteenth instructions for deciding based on said data to include more than one instance of a component into the generated document.

112. The system of claim 109, further comprising twelfth program instructions for assigning a unique identifier to the first component instance, for associating the unique identifier with the data object, and for including the unique identifier into the generated document, said twelfth program instructions for execution while dynamically generating a document .

113. The system of claim 112, wherein fifth program instructions analyze said data for unique identifiers and include thirteenth instructions for identifying the associated data object.

114. A system for displaying dynamically generated documents in a data network coupling a server computer to a client computer, wherein the client computer has a first software program including first program instructions for generating a request to obtain at least one generated document from the server computer and for displaying the generated document, comprising:

a plurality of components for execution on the server, at least one of the components including first features to cooperate with an editor in editing said component and second program instructions to generate browser code, and

third program instructions on the server for, based on the data contained in a request initiated by the client computer, generating generated documents for transfer to the client computer, thereby calling second program instructions of selected components.

115. The system of claim 114 wherein first features include fourth program instructions for passing information to the editor.

116. The system of claim 115 wherein at least part of said information is collected during execution of selected components on the server.

117. The system of claim 115 wherein said information is transmitted from the server to the client.

118. The system of claim 115 wherein said information includes attributes values of said component.

119. The system of claim 114 wherein first features include fifth instructions that display additional editing features of the component during editing.

120. The system of claim 119 wherein said editing features include handles.

121. The system of claim 114 wherein first features include an extension for use by the editor, said extension for enabling editing of an attribute value of the components.

122. The system of claim 121 wherein said extension is enables display of a page for editing the components attributes values.

123. The system of claim 114 wherein at least one component is denoted on at least one document templates using tag syntax, whereby the tag name identifies a component kind.

124. The system of claim 114 containing at least one component wherein second program instructions are used to generate browser code for displaying the component during editing and during normal use.

125. A method for editing an application that is built using components and that operates by generating documents comprising the steps of:

running the application, thereby executing selected components and generating a generated document,

displaying a view of the generated document,

selecting a component by clicking on selected portions of said view,

identifying the selected component in the source code of the application,

initiating a modification function modifying the source code of the application.

126. The method of claim 125 wherein the running step and the displaying step are repeated after applying a modification function.

127. The method of claim 125 further comprising collecting edit information for use by the identifying step.

128. The system of claim 114 additionally comprising a plurality of document templates with components denoted thereon, whereby the browser code generated by the components can vary for different requests of the same document template.

IX. EVIDENCE APPENDIX
-NONE-

X. RELATED PROCEEDINGS APPENDIX
-NONE-